Optimizing Streaming Applications with Self-Interested Users using M-DPOP

Optimizing Streaming Applications with Self-Interested Users using M-DPOP

Boi Faltings*, David Parkes†, Adrian Petcu‡, Jeff Shneidman§

Abstract

In this paper we deal with the problem of optimally placing a set of query operators in an overlay network. Each user is interested in performing a query on streaming data and each query has an associated set of in-network operators that filter, aggregate and process the data in various ways. Each user has private information about the operators associated with a query and about the utility from different combinations of operator placements. Each server in the overlay network is able to perform some set of operators, and servers differ in their network and computational characteristics.

We model this problem as a Distributed Constraint Optimization Problem (DCOP), and apply the M-DPOP algorithm from Petcu et al. [19], executed here by clients associated with users and situated at nodes on the overlay network. M-DPOP makes truth-telling an *ex-post Nash equilibrium* and determines the social-welfare maximizing placement of operators to servers. No client can benefit by deviating from the M-DPOP algorithm and nodes need only communicate with other nodes that have an interest in placing an operator on the same server. The only central authority required is a bank that can extract payments from users. Preliminary results from simulation show that message size will be a bottleneck in applying M-DPOP to operator placement unless structure can be enforced and then exploited.

1 Motivation

Recently, there has been interest in building long-lived data streaming applications on the Internet. These applications typically involve querying, processing, and delivering real-time data from multiple distributed data sources, such as sensor networks, and making use of shared resources in the Internet to aggregate, filter, or multicast this data. Commonly-cited target applications include continuous monitoring of Internet paths and system loads [8], and querying geographically diverse data sources [20].

These streaming applications can run on overlay networks that consist of nodes that are capable of performing in-network processing. A few examples of such overlays are IrisNet [7], PIER [8], Borealis [1], and SBON [20]. In these networks, *queries* are submitted by users who wish to receive data from

*boi.faltings@epfl.ch, EPFL, CH-1015 Lausanne
†parkes@eecs.harvard.edu, Harvard University, DEAS, Cambridge MA 02138, USA
‡adrian.petcu@epfl.ch, EPFL, CH-1015 Lausanne
§jeffsh@eecs.harvard.edu, Harvard University, DEAS, Cambridge MA 02138, USA

producers via one or more in-network operators. Examples of in-network operators include database style "join" operators, or custom logic provided by an end user.

One of the fundamental questions in these overlay networks is how to perform *query placement*, which is the problem of mapping the *operators* in a particular query to a collection of nodes (the *servers*) that will run those operators. Placement can be formulated as a complex constrained optimization problem; placements must be done subject to load and bandwidth node limitations (where important), quality of service stream requirements, and done in a computationally efficient way.

In practice, users have varying levels of happiness, or *utility*, for how their queries are placed and executed. For example, a user with a jitter-sensitive query should avoid operator placement on nodes with high variance in communication latency. On the other hand, a user with a high volume query may not care about latency but wants operators placed on nodes with high data rates.

The goal of our research is to provide a distributed constrained optimization algorithm that allows users to influence where their query operators are placed, but seeks to maximize the aggregate utility across all users. This approach is unique in that it is the first overlay query placement algorithm that uses ideas from computational mechanism design to compute a value-maximizing placement in the presence of self-interested nodes. Our solution leverages M-DPOP [19], which provides a distributed algorithm for social choice problems, and can exploit problem structure to scale well to large problems. M-DPOP is a *faithful* distributed implementation [21], in the sense that even nodes controlled by users (and thus open to manipulation) will choose to follow the algorithm because this maximizes, in equilibrium, their individual self-interest. Preliminary results from simulation show that message size will be a bottleneck in applying M-DPOP to operator placement unless structure can be enforced and then exploited.

2 Problem Statement

Each user is associated with a query and has a client located at a particular node on the overlay network. Each query has an associated set of data producers, known to the user and located at nodes on the network. Each query also requires a set of operators, to be placed on (server) nodes between the producers and the user's node. Each user assigns values (or *utilities*) to various allocations of operators to servers. This preference information is private, and users are assumed self-interested and seek to maximize their individual utility.

We seek a distributed algorithm, to be executed by user clients situated on network nodes, that will determine the allocation and also payments to be made by each user for the outcome. The server nodes (i.e. the nodes that finally execute the operators and respond to queries) are assumed to "opt-in" in that

they will implement whatever allocation is determined by users. Constraints on server nodes, e.g. based on maximal load, are commonly known to users and thus server nodes are represented in the decision procedure. However, server nodes play no active role in the algorithm.

More formally, and adopting the term "agent" to represent a user and "utility" to describe user values, we have:

- agents $A = \{A_1, \ldots, A_n\}$, each with a query
- server nodes $B = \{B_1, \ldots, B_m\} \cup \phi$ where ϕ is the "null" node (corresponds to *operator not assigned*)
- each agent A_i has k_i operators $G_i = \{g_1, \ldots, g_{k_i}\}$ and each must be assigned to one node in the network (perhaps the null node, i.e. *not assigned*)
- each agent determines a possibility set $P(A_i, g) \subseteq B$ for each operator $g \in G_i$, which is the set of nodes for which the agent *could* have non-zero utility for placement (i.e. in some combination with other placements)
- each agent has a utility on allocations, with $u_i(x) \in \mathbb{R}$, where $x : \{G_1, \ldots, G_n\} \rightarrow B$ defines an allocation of operators to the network. (The utility is $-\infty$ if an operator is allocated to a node outside of the possibility set.)
- constraints to restrict the set of feasible allocations due to load and bandwidth considerations.

We assume there is no collusion between users and that each user controls only one client, namely its own client, and does not control any other functionality on any other nodes.

Each operator is associated with an input data stream and an output data stream. For instance, operators that process raw data received from one or more producers associated with a query define the set of producers. Thus, information about producers is implicit in the set of operators G_i associated with a query. Server nodes may be capable of running query operators on behalf of multiple concurrent queries, even allowing results of a particular operator to be re-used and shared across multiple relevant queries [20]; e.g. when the same aggregation from two producers is required by two different queries. Operator semantics must be rich enough to allow these synergies to be identified. Notice that the *null* node allows for the operator placement to decide to block one or more queries completely, or drop some of the operators in a query completely, when this is in the joint interest of all users.

Users can determine a *possibility set* of nodes for the placement of each operator. This excludes only those nodes for which the user can be absolutely certain that there is no value (to the user) for placing the operator on that

node.[5] We assume that user clients can communicate without interference with user clients representing queries that share a possible interest in a server. Each user has a utility function, which describes her value for different assignments of operators to nodes. We shall assume that a user's utility encodes bandwidth and latency considerations, and furthermore, a user is able to access enough information to determine this utility information. For instance, previous work has suggested the efficacy of various techniques to perform bandwidth and latency [22, 20] estimation between pairs of nodes [10].

The main constraint in placing the operators is that each node can only handle a limited number of operators, due to CPU and bandwidth limitations.

3 Background and Related Work

This work draws from 3 different research areas: distributed stream placement optimization, distributed constraint optimization, and faithful implementations of social choice functions.

3.1 Distributed Stream Placement Optimization

Distributed stream processing systems (DSPS) need to find a good placement for the in-network operators required by user queries. Some systems relegate this placement task back to the user. Others perform automated optimization for a hard-coded node or network metric. For instance, Borealis [1] and GATES [4] require the user to specify the initial operator locations. This forces the user to perform any optimization off-line before specifying locations. Other DSPSs, such as Medusa [3], place operators to improve application performance by balancing node load. Still other DSPs effectively randomize placement, as in PIER [8]. Neither placement strategy may be appropriate for jitter or latency-sensitive queries. SAND [2], an extension to Borealis, performs network-aware operator placement to minimize bandwidth of a query. SAND also allows applications to specify delay constraints on the query, which affects the placement decision. SBON [20] is also a system that performs network-aware operator placement, minimizing a network usage metric.

None of these previous works have taken a value-maximization (or preference-based) approach in choosing where to place operators. Rather, the system assumed that all queries were equally important, and in many cases, that queries were only concerned with latency or node load.

3.2 Distributed Constraint Optimization Framework

Distributed Constraint Optimization (DCOP), e.g. Modi et al. [11], can model social choice problems where a set of self interested agents with private utility

[5] A more advanced implementation would allow a user to state general utility functions for types of nodes – such as $u(bandwidth)$, $u(latency)$, and rely on the network to calculate the candidate set and derive the utility of each node.

functions have to agree on a set of decisions (see Petcu et al. [19]). Each decision is modeled as a variable that can take values in a well-defined domain, subject to side constraints. The goal is to maximize the total utility across all agents. Among many algorithms for this type of problems, we mention ADOPT ([11]) and DPOP ([15]). We define the general DCOP framework here. Later, in Section 4 we will instantiate operator placement in the DCOP formalism.

Definition 1 (DCOP) *A distributed constraint optimization problem (DCOP) is a tuple* $< \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R} >$ *such that:*

$\mathcal{A} = \{A_1, ..., A_n\}$ *is a set of* **self-interested** *agents interested in the optimization problem;*

$\mathcal{X} = \{X_1, ..., X_m\}$ *is the set of* **public** *decision variables;* $X(A_i)$ *are the variables in which agent* A_i *is interested and* ***does*** *have relations. Agents* A_i *for which* $X_j \in X(A_i)$ *for some variable* X_j *form the* community *for variable* X_j*, denoted* $A(X_j) \subseteq \mathcal{A}$*.*

$\mathcal{D} = \{d_1, ..., d_m\}$ *is the set of finite* **public** *domains of the variables* $\mathcal{X}$*; each domain* d_i *is known to all interested agents (i.e. agents* $A_j s.t. X_i \in X(A_j)$*;*

$\mathcal{C} = \{c_1, ..., c_q\}$ *is a set of* **public** *constraints, where a constraint* c_i *is a function* $c_i : d_{i_1} \times .. \times d_{i_k} \rightarrow \{-\infty, 0\}$ *that returns 0 for all allowed combinations of values of the involved variables, and* $-\infty$ *for disallowed ones; these constraints are known and agreed upon by all agents involved in the respective communities;*

$\mathcal{R} = \{R_1, ..., R_n\}$ *is a set of* **private** *relations, where* R_i *is the set of relations specified by agent* A_i *and relation* $r_i^j \in R_i$ *is a function* $d_{j_1} \times .. \times d_{j_k} \rightarrow \mathbb{R}$ *specified by agent* A_i*, which denotes the utility* A_i *receives for all possible values on the involved variables* $\{j_1, \ldots, j_k\}$ *(negative values can be thought of as costs). An agent's utility for a complete assignment of values to variables is defined by the sum of its relations.*

The optimal solution is a complete instantiation X^* *of all variables in* $\mathcal{X}$*, s.t.* $X^* = argmax_{X \in \mathcal{D}}(\sum_{R_i \in \mathcal{R}} R_i(X) + \sum_{c_i \in \mathcal{C}} c_i(X))$,[6] *where* $R_i(X) = \sum_{r_i^j \in R_i} r_i^j(X)$ *is* A_i*'s utility for this solution.*

Later, we use DCOP($-A_i$) to denote the constraint optimization problem without agent A_i, and refer to this as the "marginal problem without agent A_i."

In addition to private relations on public variables, DCOP allows an agent to have *private variables* and arbitrary relations and constraints imposed on subsets of private variables and public variables. Decisions about private variables, as well as explicit information about these relations and constraints remain private to an agent.

In our context, variables will be associated with each instance of an operator, and domains with the servers that are of possible interest to the user associated

[6]Notice that the second sum is either $-\infty$ if X is an infeasible assignment, or 0 if it is feasible. Thus, optimal solution X^* will always satisfy all hard constraints when that is possible.

with the operator. Relations provide a method to express factored utilities, with the utility for an allocation decomposed into an aggregate over utilities for server assignments on groups of operators that are inputs and outputs to each other.

3.3 M-DPOP: faithful utilitarian social choice

Petcu et al. [19] have proposed *M-DPOP*, a distributed optimization protocol that *faithfully* implements (in the sense of Shneidman and Parkes [21]) the Vickrey-Clarke-Groves (VCG) mechanism ([6]) for the problem of utilitarian social choice. No agent can benefit by unilaterally deviating from any aspect of the protocol, neither information-revelation, computation, nor communication. Additionally, M-DPOP provides a faithful method to redistribute some of the VCG payments back to agents (weak budget-balance). The optimization algorithm itself is based on *DPOP* ([15]), which is a dynamic programming algorithm adapted for distributed constraint optimization problems. Agents need only communicate with other agents that have an interest in the same variable, and provided that DPOP scales then the entire method of M-DPOP scales.

Briefly, M-DPOP has the following phases:

1. *Community formation and DFS creation:* the agents interested in the value of a variable X_i organize themselves in the *community* $A(X_i)$ of that variable (a community can be physically implemented as a public medium like a bulletin board, a mailing list, etc.) All agents interested in X_i subscribe to X_i's community. Each agent $A_i \in A(X_i)$ then creates its own replica of X_i, and expresses its preferences on combinations of X_i and other variables as local relations on the local copies of these variables. By doing so, each agent creates its *local optimization problem*, denoted $COP(A_i)$. Copies of the private variables are synchronized among all interested agents using equality constraints. Once the constraint graph is established, a depth-first-search (DFS) traversal is constructed starting from a randomly chosen node. This defines the control logic.

2. *Solving the main problem:* DPOP is run on the previously established DFS structure, and the optimal solution for $DCOP(\mathcal{A})$ is obtained. DPOP involves a bottom-up propagation (and aggregation) of utility information followed by a top-down propagation of assignment information.

3. *Solving each marginal problem:* DPOP is then run in parallel on each marginal problem. Computation from the main problem (i.e. residual local state) is reused for solving each marginal problem, $DCOP(-A_i)$, in a way that prevents manipulation by A_i. Finally, the VCG taxes are then computed distributedly again in a non-manipulable fashion by all agents except the one whose tax is computed, and levied by a trusted bank.

M-DPOP's complexity in terms of number of messages is always linear in the number of variables in the optimization problem. In terms of message size, the largest message sent by any agent while executing M-DPOP is $O(exp(w))$, where w the induced width of the constraint graph ([5]). Roughly, a small induced width reflects problems with limited interconnectedness between decisions.

4 DCOP Models for Optimal Operator Placement

The problem structure of an instance of DCOP($\mathcal{A}$) can be represented as a *multigraph*, with the decision variables as nodes, and (possibly) multiple relations belonging to different agents that involve the same variables, and expressing their utilities. Figure 1(a) shows an example where the variables are associated with servers, domains are combinations of operators, and agents express preferences on combinations thereof, in the form of constraints and relations on those variables.

We adopt an alternate formulation, depicted in Figure 1(b)), in which the variables are associated with individual operators and the domains are servers of possible interest for an operator.[7] In order to allow multiple agents to express preferences on the same set of variables, we require distributed models where each agent can model its own interests as an internal optimization problem ($COP(A_i)$), and interactions between agents (agreement, feasibility constraints) are modeled as inter-agent constraints.[8] This is reflected in the model of operator placement.

4.1 DPOP model for Operator Placement

4.1.1 Local optimization problem

The local optimization problem $COP(A_i)$ of agent A_i models A_i's interests and is composed of private **variables** and **relations** (see Figure 1(b) for an example). Each agent A_i creates one variable $A_i g_j$ for each one of its operators g_j. The domain of a variable $A_i g_j$ is the possibility set $P(A_i, g_j)$ for the agent-operator (A_i, g_j) to whom the variable relates. These variables are private to agents and each agent has as many variables as it has operators to assign.

Each agent has *relations* on the values assigned to its variables (blue edges in Figure 1). These relations may be factored, e.g. perhaps one operator must be placed on any of some set of nodes with particular properties (Linux, high-bandwidth, etc.) while the other two operators should be within 3 hops of each other. Intra-agent constraints (i.e. private) may constrain combinations of operator positions that are not suitable to the agent. For example, in Figure 1),

[7] A subtle incentive problem exists with the servers-as-variables model which will be explained in a longer version of this paper.

[8] Local, private variables do not show up in inter-agent communication and agents typically need not solve the internal problem for all combinations of values of the public variables [23].

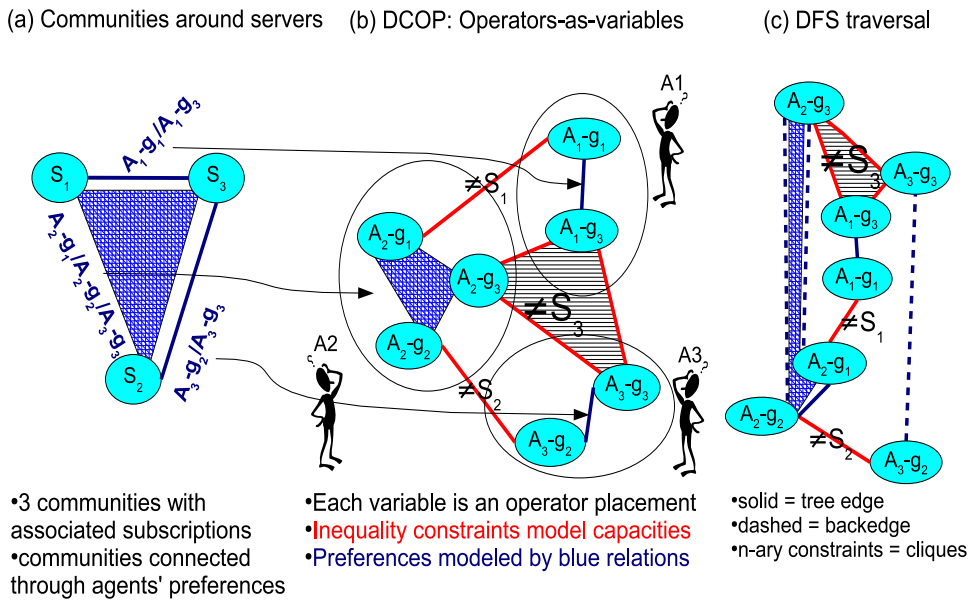

Figure 1: *An operator placement problem: (a) community formation , (b) DCOP model (operators-as-variables), and (c) DFS arrangement*

A_2 could express with its ternary relation (blue area connecting all its variables) that it prefers to have all its operators assigned on machines with the same OS, or that the sum of the bandwidths of the hosting servers must exceed a certain threshold, etc.

4.1.2 Interdependencies between local optimization problems

Local optimization problems are connected through *interagent* constraints. In this case, interagent constraints (commonly known to all agents) represent capacity constraints. For instance, consider a particular node h and let $X(h)$ denote the variables that include h in their domain. For each such h, there could be a constraint of the form "no more than C_h (some small integer) unique operators can be assigned to this node." For example, in Figure 1, agents A_1, A_2, and A_3 are each interested in placing an operator ($A_1\_g_3$, $A_2\_g_3$ and $A_3\_g_3$, respectively) on S_3. Since S_3 has limited capacity, all three variables are connected through a ternary capacity constraint. *Note that synergies can be found between operators in this formulation since agents 1 and 2 may have the same operator (e.g. apply an aggregation operator to readings from the same two producer nodes), and thus the coordinated decision by both agents to place this operator on the same node would only count once against the capacity of that node.* Agents must be able to identify this equivalence between operators.

4.2 Applying M-DPOP to Operator Placement

We describe in more detail the initial phase of M-DPOP whereby the communities are formed and the DFS structure is constructed. As user's variables in this model are initially private the process is slightly changed from that in Petcu et al. [19]:

1. Each agent A_i expresses internally its interests as an optimization problem $COP(A_i)$ (see 4.1.1)
2. Agents subscribe to the servers where they would like to place operators (see Figure 1). Each server S_j maintains a public subscriber list $A(S_j)$, and at the end of the subscription process, notifies all subscribers. Agents $A(S_j)$ are referred to as the *community* of server S_j. Each subscriber connects its corresponding variable to all other variables, thus forming a clique that corresponds to an n-ary capacity constraint.
3. Every agent can infer (or the server can specify) what combinations of operators observe the capacity of the server and capture this information via hard constraints.
4. Once the constraint graph is thus established, a depth-first-search traversal is constructed starting from a randomly chosen node (see Figure 1(c) for an example DFS)

Next, the bottom-to-top utility propagation and top-to-bottom decision propagation phases proceed as in M-DPOP. The capacity constraints from this model are the equivalent of the *hard-constraints* in M-DPOP parlance. As in M-DPOP, they are treated by the lowest agent in the DFS tree that has a variable involved in the constraint. For example, in Figure 1(c), we have the capacity constraint corresponding to S_3 as the shaded area involving $A_1\_g_3$, $A_2\_g_3$ and $A_3\_g_3$. This would be handled by A_1 when it sends its *UTIL* message from $A_1\_g_3$ to $A_3\_g_3$. A_1 does this by assigning $-\infty$ to all value combinations of $A_1\_g_3$, $A_2\_g_3$ and $A_3\_g_3$ that violate the S_3 capacity constraint, and the normally computed valuations to the other combinations.

This ensures that throughout the whole propagation, all combinations of operator assignments that violate at least one capacity constraint will be assigned $-\infty$ utility and will therefore be avoided. On termination, once DCOP has been run for the main and each marginal problem the solution to the main problem is adopted by the server nodes and the bank collects VCG payments (defined in terms of the difference between main and marginal problem solutions and reported in a distributed manner.)

Applying M-DPOP to this domain provides a protocol that is *ex post Nash* faithful, meaning that if each client follows the M-DPOP algorithm then no single client can benefit by deviating from the algorithm (including in reporting untruthful information) *whatever* the private relations of the other agents.[9]

[9] In game-theoretic terms, the algorithm prescribes an *ex post* Nash equilibrium. The usual dominant-strategy equilibrium property that is achieved in VCG mechanisms is weakened to

5 Scalability of M-DPOP

This section presents a theoretical complexity analysis of our algorithm (Section 5.1), and an experimental evaluation on randomly generated problems (Section 5.2).

5.1 Theoretical Complexity

At the core of M-DPOP([19]), we use the DPOP([15]) algorithm for constraint optimization. Consequently, complexity-wise, M-DPOP can be seen as a sequence of DPOP runs: one for the main economy, and then another one for each marginal economy. M-DPOP has a mechanism for identifying parts of the computation from the main economy that can be safely (manipulation-free) reused in each marginal economy. Therefore, only in the worst case, when no effort can be reused, M-DPOP requires n+1 times the effort spent by DPOP. For more detail, please refer to Petcu et al. [19].

DPOP's complexity can be specified along two dimensions. First, in terms of number of messages, DPOP has the advantage that it requires only a linear number of messages. Second, in terms of the size of the messages, DPOP produces in the worst case messages whose size depends on the structure of the problem graph. This structure is captured with a parameter called the *induced width* of the graph, which depends on the clustering and the connectedness of the graph. It is important to notice that the width of the graph does not depend directly on the size of the graph, which means that certain types of problems can be easily solved although they are very large. Specifically, large but loose problems can be solved efficiently with DPOP. For more detail and formal proofs, please refer to Petcu and Faltings [15].

5.2 Experimental Evaluation

It is well known that the VCG mechanism requires optimal solutions in order to guarantee faithfulness. When applied to hard problems, this can render such schemes unfeasible, because finding the optimal solutions may be computationally impossible.

We present results from a simulation study to understand the scalability of our algorithm for solving operator placement problems in a multi-user, multi-server environment. We explore the relationship between the algorithm's computational and communication complexity and the number of agents and servers in the system. All results were generated using the FRODO multiagent simulation platform [13].

Users are divided into *similarity classes* so that users in different classes have (mainly) non-overlapping interest in different operators. This models a real overlay, where different classes of users issue non-overlapping queries.

ex post Nash: for example, truth-revelation is only a best-response if the other agents choose to implement the rules of the VCG mechanism correctly (which they will in equilibrium.)

Agents	Srvs	Vars	Constr	Msgs	MaxMsgSize	TotalMsgSize
20	10	32	42	31	96896	403399
40	19	60	78	59	117248	386314
60	29	92	117	91	303104	458533
80	40	128	171	127	1255424	2295570
100	49	156	210	155	1128675	2966304
120	59	188	255	187	20067123	33897613

Table 1: M-DPOP experiments on operator placement problems.

In these experiments, the classes are generated and then around 10 random users are added to a class. Classes are grouped in a hierarchy, according to similarity criteria at the class level. Users from neighboring (similar) classes can sometimes issue queries that cross class boundaries. Each user generates a query consisting of a random subset of operators from this user's similarity class (high probability) or another similar class (low probability). Each query consists of between one and five operators, which is consistent with previous work on stream queries [20]. Servers with random capabilities are then generated: each server is able to execute some random subset of the union of operators from a set of neighboring similarity classes. For each operator, the user generates a random *possibility set*, as described in Section 2. The possibility set for each operator is limited, since each server is allowed to run only a subset of the available operators. This models a real overlay where operators may require servers with special capabilities, memory, operating systems, etc. The user assigns random valuations for combinations of operator placements onto servers from each operators' possibility set. With all utility assignments made, the users then run M-DPOP.

Table 1 shows how our algorithm scales up with the size of the problems. The columns mean, in order: number of users in the network, number of servers, number of variables in the resulting problem, number of constraints in the resulting problem, number of messages required by DPOP to solve the problem, the size of the largest UTIL message in DPOP, and the total size of the UTIL messages generated in one solving process. The unit of the message size is valuations; in DPOP terms, it is the number of valuations of a UTIL message. The maximum message size is the size (in valuations) of the largest message, and the total is the summed size (in valuations) of all messages.

This table shows an explosion in message size as the problem gets large. This explosion is caused by two factors: the domain size of the variables, and the width of the DFS structure used by the M-DPOP algorithm. The width is adversely affected by the overlap of many queries on the same possible servers that could execute them. In terms of our operator placement problem, one can increase the scalability if one can make the similarity classes smaller and more disjoint and the possibility sets more disjoint. We plan to investigate how

these conditions can be achieved in practice in order to achieve good scalability, for instance by imposing constraints on the problem that will enforce sufficient structure.

6 Discussion

6.1 Dynamic allocation problems

The streaming application problem is really a dynamic problem. For dynamically evolving environments, Petcu and Faltings proposed in [17] a self-stabilizing version of DPOP that is guaranteed to continuously follow the evolution of a dynamic problem, always finding the optimal solution. An extended version of this technique ([16]) also ensures that when a new solution is derived upon a change, the cost of revising previously taken decisions is also taken into account.

One can leverage these techniques for the purpose of optimizing streaming applications as well. An important requirement is that the rate of change in the environment is small enough to allow the algorithm to stabilize and find the optimal solution. Provided this is the case, one can simply regard this evolution as a sequence of M-DPOP executions, where computation can be reused from $DCOP(\mathcal{A}, t_k)$ to $DCOP(\mathcal{A}, t_{k+1})$, and from $DCOP(-A_i, t_k)$ to $DCOP(-A_i, t_{k+1})$. Optimal solutions are computed, and taxes are levied once per time period. The ex post faithfulness properties are retained as long as user utilities can be decomposed in a linear fashion across time periods (e.g., when query streams are interruptible and utility accrues for each period of time a stream of a particular quality is received.) However, it will be important to understand whether new, undesirable equilibria are introduced in moving to the multi-period setting.

6.2 Approximations

M-DPOP is a complete algorithm (in the AI sense) and is guaranteed to terminate with the optimal solution. However, this guarantee comes at the cost of potentially large messages sizes. A practical systems solution must avoid such worst-case behavior. In earlier work, Petcu and Faltings [14] have shown that approximations of dramatically lower complexity still provide results that can be expected to be quite close to the optimum. The challenge in adopting approximate solutions within the framework of M-DPOP, and thus mechanism design, is that approximations can cause the *faithfulness* and incentives for truthfulness to unravel. The VCG payments continue to provide "self-correcting" incentives even with approximations (see Nisan and Ronen for example [12]), but progress in identifying useful equilibrium concepts in this context remains an important open problem in computational mechanism design. One way to retain faithfulness while introducing approximations is to impose constraints on the space of solutions that will be considered, and in a

way that is fixed and independent of agent messages. Progress in this direction remains future work.

7 Concluding Remarks

We have presented a distributed optimization approach to the optimal operator placement problem. We have introduced a DCOP model for this problem, and showed how one can apply M-DPOP to these problems. M-DPOP is a recently introduced optimization algorithm that makes faithful execution an *ex-post* Nash equilibrium. As M-DPOP is a derivative of DPOP, various techniques like self-stabilization for dynamic systems (see [17]) or linear size messages (see [18]) can be applied.

Preliminary results from simulation demonstrate that it will be of critical importance in large problem instance to identify, and then leverage, useful problem structure. We believe, for example, that one can take advantage of the special structure of the capacity constraints to develop more computationally efficient techniques, like Kumar et al. [9]; we will investigate these avenues in future work.

References

[1] D. Abadi, Y. Ahmad, H. Balakrishnan, et al. The Design of the Borealis Stream Processing Engine. Technical Report CS-04-08, Brown University, July 2004.

[2] Y. Ahmad and U. Çetintemel. Network-Aware Query Processing for Stream-based Applications. In *VLDB*, Aug. 2004.

[3] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-Based Load Management in Federated Distributed Systems. In *Proc. of NSDI'04*, San Francisco, CA, Mar. 2004.

[4] L. Chen, K. Reddy, and G. Agrawal. GATES: A Grid-Based Middleware for Processing Distributed Data Streams. In *Proc. of HPDC*, June 2004.

[5] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[6] E. Ephrati and J. Rosenschein. The Clarke tax as a consensus mechanism among automated agents. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-91*, pages 173–178, Anaheim, CA, July 1991.

[7] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: An Architecture for a World-Wide Sensor Web. *IEEE Pervasive Computing*, 2(4), Oct. 2003.

[8] R. Huebsch, J. M. Hellerstein, N. Lanham, et al. Querying the Internet with PIER. In *VLDB*, Sept. 2003.

[9] A. Kumar, A. Petcu, and B. Faltings. H-DPOP: Using hard constraints to prune the search space. In *IJCAI'07 - Distributed Constraint Reasoning workshop, DCR'07*, Hyderabad, India, Jan 2007.

[10] K. Lai and M. Baker. Measuring bandwidth. In *INFOCOM*, pages 235–245, 1999.

[11] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *AI Journal*, 161:149–180, 2005.

[12] N. Nisan and A. Ronen. Computationally feasible VCG mechanisms. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, pages 242–252, New York, NY, USA, 2000. ACM Press.

[13] A. Petcu. FRODO: A FRamework for Open/Distributed constraint Optimization. Technical Report No. 2006/001, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, 2006. http://liawww.epfl.ch/frodo/.

[14] A. Petcu and B. Faltings. Approximations in distributed optimization. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP'05)*, Sitges, Spain, October 2005.

[15] A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI-05*, Edinburgh, Scotland, Aug 2005.

[16] A. Petcu and B. Faltings. Optimal solution stability in continuous time optimization. In *IJCAI05 - Distributed Constraint Reasoning workshop, DCR05*, Edinburgh, Scotland, August 2005.

[17] A. Petcu and B. Faltings. S-DPOP: Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-05*, Pittsburgh, USA, July 2005.

[18] A. Petcu and B. Faltings. O-DPOP: An algorithm for Open/Distributed Constraint Optimization. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-06*, Boston, USA, July 2006.

[19] A. Petcu, B. Faltings, and D. Parkes. M-DPOP: Faithful Distributed Implementation of Efficient Social Choice Problems. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-06)*, Hakodate, Japan, May 2006.

[20] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-Aware Operator Placement for Stream-Processing Systems. In *ICDE*, April 2006.

[21] J. Shneidman and D. C. Parkes. Specification faithfulness in networks with rational nodes. In *Proc. of the 23rd ACM Symposium on Principles of Distributed Computing (PODC'04)*, St. John's, Canada, 2004.

[22] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: a lightweight network location service without virtual coordinates. In *SIGCOMM*, pages 85–96, New York, NY, USA, 2005. ACM Press.

[23] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem - formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.